Aufgabe 1: Ankreuzfragen (30 Punkte)

1) Einfachauswahlfragen (22 Punkte)

Bei den Einfachauswahlfragen in dieser Aufgabe ist jeweils nur **eine** richtige Antwort eindeutig anzukreuzen. Auf die richtige Antwort gibt es die angegebene Punktzahl.

Wollen Sie eine Antwort korrigieren, streichen Sie bitte die falsche Antwort mit drei waagrechten Strichen durch (☒) und kreuzen die richtige an.

Lesen Sie die Frage genau, bevor Sie antworten.

a) Welche der folgenden Aussagen zum Thema Prozesszustände ist richtig? 2 Punkte

- ☐ Bei Eintreffen eines Interrupts wird der aktuell laufende Prozess für die Dauer der Interrupt-Abarbeitung in den Zustand blockiert überführt.
- ☐ Im Rahmen der mittelfristigen Einplanung kann ein Prozess von Zustand laufend in den Zustand schwebend laufend wechseln.
- ☐ Das Auftreten eines Seitenfehlers kann dazu führen, dass der aktuell laufende Prozess in den Zustand beendet überführt wird.
- ☐ Bei kooperativem Scheduling ist kein direkter Übergang vom Zustand laufend in den Zustand bereit möglich.

b) Welche Aussage zur Seitenumlagerung in virtuellen Adressräumen ist richtig? 2 Punkte

- ☐ Unter Seitenflattern versteht man das ständige Ein- und Auslagern von Speicherseiten, wenn der physisch vorhandene Hauptspeicher nicht ausreicht.
- ☐ Bei der Seitenersetzungsstrategie LRU wird diejenige Seite ausgelagert, auf die in der Vergangenheit am seltensten zugegriffen wurde.
- ☐ Beim Auslagern einer Speicherseite muss der zugehörige Seitendeskriptor angepasst werden. Beim Einlagern ist dies nicht nötig.
- ☐ Die Seitenersetzungsstrategie Second Chance (Clock) ist nur in der Theorie interessant, weil ihre Implementierung komplexe Datenstrukturen erfordert.

c) Beim Blockieren in einem Monitor muss der Monitor freigegeben werden. Warum? 2 Punkte

- ☐ Weil kritische Abschnitte immer nur kurz belegt sein dürfen.
- ☐ Weil sonst die Monitordaten inkonsistent sind.
- ☐ Weil ein anderer Thread die Blockierungsbedingung nur aufheben kann, wenn er den Monitor vorher betreten kann.
- ☐ Weil der Thread sonst aktiv warten würde.

h) Was ist ein Stack-Frame? 2 Punkte

- ☐ Der Speicherbereich, in dem der Programmcode einer Funktion abgelegt ist.
- ☐ Ein Fehler, der bei unberechtigten Zugriffen auf den Stack-Speicher entsteht.
- ☐ Ein Bereich des Speichers, in dem u.a. lokale automatic-Variablen einer Funktion abgelegt sind.
- ☐ Ein spezieller Registersatz des Prozessors zur Bearbeitung von Funktionen.

i) Welche der folgenden Informationen wird typischerweise in dem Seitendeskriptor einer Seite eines virtuellen Adressraums gehalten? 2 Punkte

- ☐ Die Zugriffsrechte auf die jeweilige Seite (z. B. lesen, schreiben, ausführen).
- ☐ Die Identifikation des Prozesses, dem die Seite zugeordnet ist.
- ☐ Die Zuordnung zu einem Segment (Text, Daten, ...).
- ☐ Die Position der Seite im virtuellen Adressraum.

j) Welche der folgenden Aussagen zu statischem bzw. dynamischem Binden ist richtig? 2 Punkte

- ☐ Änderungen am Code einer dynamischen Bibliothek (z. B. Bugfixes) erfordern immer das erneute Binden aller Programme, die diese Bibliothek benutzen.
- ☐ Beim statischen Binden werden alle Adressen zum Ladezeitpunkt aufgelöst.
- ☐ Beim dynamischen Binden erfolgt die Adressauflösung beim Laden des Programms oder zur Laufzeit.
- ☐ Statisch gebundene Programme können zum Ladezeitpunkt an beliebige virtuelle Speicheradressen platziert werden.

k) Sie kennen den Translation-Lookaside-Buffer (TLB). Welche Aussage ist richtig? 2 Punkte

- ☐ Verändert sich die Speicherabbildung von logischen auf physikalische Adressen aufgrund einer Adressraumumschaltung, so werden auch die Daten im TLB ungültig.
- ☐ Der TLB verkürzt die Zugriffszeit auf den physikalischen Speicher da ein Teil des möglichen Speichers in einem schnellen Pufferspeicher vorgehalten wird.
- ☐ Der TLB puffert Daten bei der Ein-/Ausgabebehandlung und beschleunigt diese damit.
- ☐ Wird eine Speicherabbildung im TLB nicht gefunden, wird der auf den Speicher zugreifende Prozess mit einer Schutzraumverletzung (*Segmentation Fault*) abgebrochen.

b) Man unterscheidet zwei Kategorien von Ausnahmesituationen bei einer Programmausführung: Traps und Interrupts. Welche der folgenden Aussagen sind zutreffend?

4 Punkte

- ◯ Die CPU sichert bei einem Interrupt einen Teil des aktuellen Prozessorzustands.
- ◯ Ein Systemaufruf im Anwendungsprogramm ist der Kategorie Interrupt zuzuordnen.
- ◯ Bei einem Trap wird der gerade in Bearbeitung befindliche Maschinenbefehl immer noch vollständig zu Ende bearbeitet, bevor mit der Trapbehandlung begonnen wird.
- ◯ Da Traps immer synchron auftreten, kann es im Rahmen ihrer Behandlung nicht zu Wettlaufsituationen mit dem unterbrochenen Programm kommen.
- ◯ Wenn ein Interrupt einen Fehler signalisiert, wird das unterbrochene Programm im Rahmen der Interruptbearbeitung immer abgebrochen.
- ◯ Ein durch einen Interrupt unterbrochenes Programm darf je nach der Interruptursache entweder abgebrochen oder fortgesetzt werden.
- ◯ Ein Programm darf im Rahmen einer Trapbehandlung abgebrochen werden.
- ◯ Das Dereferenzieren eines Zeigers kann zu einem Trap führen.

Sie dürfen diese Seite zur besseren Übersicht bei der Programmierung heraustrennen!

```c
#include <stdio.h>
#include <stdbool.h>
#include <signal.h>
#include <errno.h>
#include <pthread.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <string.h>
#include <netinet/in.h>
#include <unistd.h>
#include <limits.h>

#include "bbuffer.h" // Vorgegeben, siehe Manpage.

// Schnittstelle des vorgegebenen Semaphor-Moduls:
typedef struct SEM SEM;
SEM *semCreate(int initVal);
void semDestroy(SEM *sem);
void P(SEM *sem); // Decrement.
void V(SEM *sem); // Increment.

static const size_t   NUMTHR = 8;
static const size_t   BBSIZE = 8;
static const uint16_t ADMIN_PORT = 2222, CLIENT_PORT = 8888;
static const int      NEWMSG_PILL = -1, SIGINT_PILL = -2;
#define               LMAX 1395

// Vorgegeben Strukturen:
typedef struct msg {
  int id;
  char cont[LMAX + 1];
} msg_t;

// Vorgegebene Funktionen:
static void die(const char *msg) {
  perror(msg);
  exit(EXIT_FAILURE);
}

static int parse_positive_int(char *str) {
    /* Gibt einen positiven int zurück, oder -1 im Fehlerfall. */
}

static void dummy_connect(uint16_t port) {
    /* Sorgt dafür, dass ein auf port blockierter accept() Aufruf zurückkehrt
     * indem eine Verbindung aufgebaut, und sofort wieder geschlossen wird. */
}
```

```
  // Fäden starten:
```

```
  // Passives Warten auf SIGINT:
```

```
  // Geregeltes Aufräumen:
```

```
  // Nächste Seite: "Auf Arbeiterthreads warten und
  //   Ressourcen freigeben"
```

```
static void spawn(pthread_t *tid, void *(*routine)(void *)) {

}
```

P:

```
static void *accept_admins(void *arg) {

}
```

A:

```
static void *accept_clients(void *arg) {
```

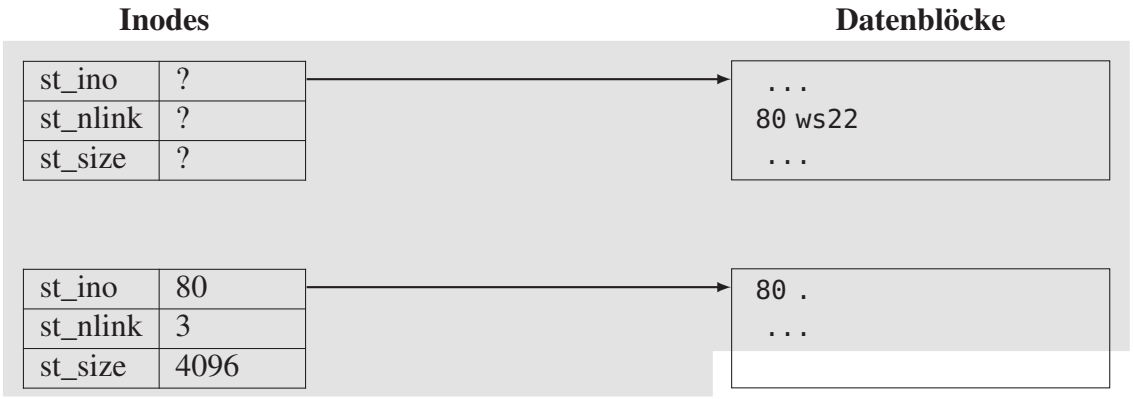
```
}
```

C:

```
static void *serve_clients(void *arg) {
```

```
}
```

S:



st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

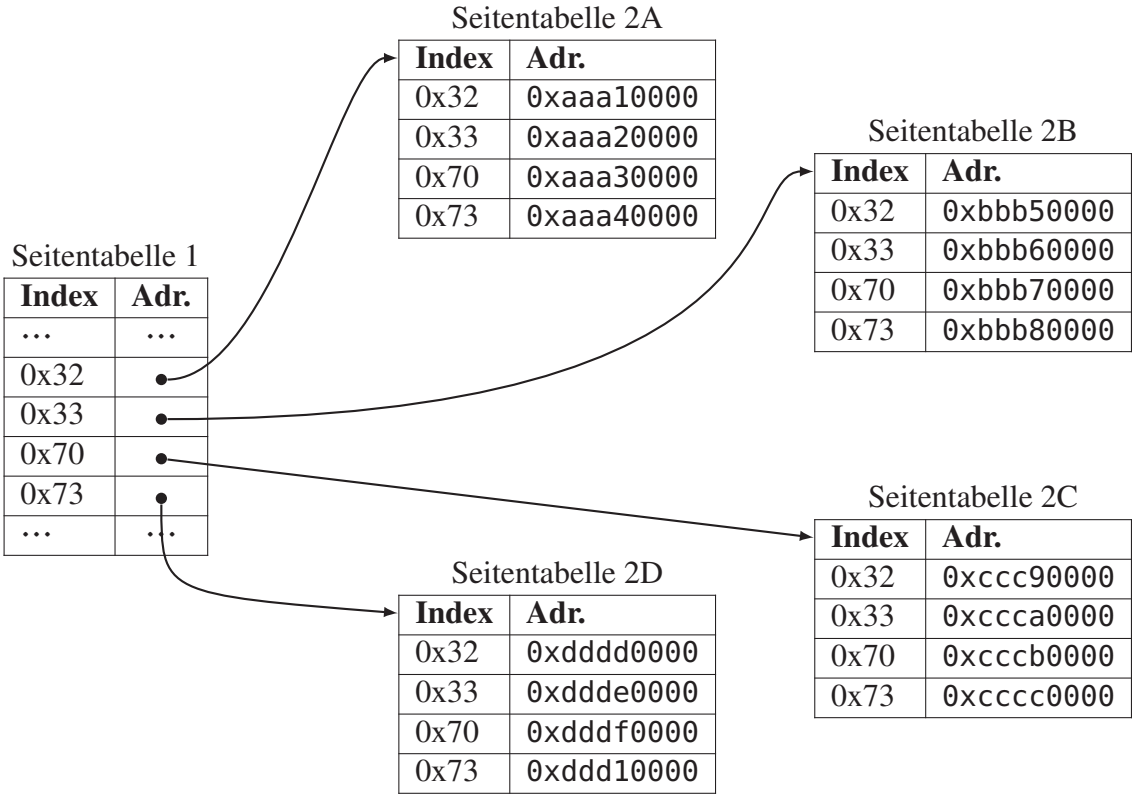
st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

st_ino	
st_nlink	
st_size	

2) Gegeben sei unten dargestellte Hierarchie zweistufiger Seitentabellen. Die Adresslänge des genutzten Systems sei 32 Bit, die Größe einer Seite 64 kByte. Für die Indizierung der zweistufigen Abbildung werden pro Stufe 8 Bit genutzt. (7 Punkte)



Seitentabelle 1

Index	Adr.
…	…
0x32	
0x33	
0x70	
0x73	
…	…

Seitentabelle 2A

Index	Adr.
0x32	0xaaa10000
0x33	0xaaa20000
0x70	0xaaa30000
0x73	0xaaa40000

Seitentabelle 2B

Index	Adr.
0x32	0xbbb50000
0x33	0xbbb60000
0x70	0xbbb70000
0x73	0xbbb80000

Seitentabelle 2C

Index	Adr.
0x32	0xccc90000
0x33	0xccca0000
0x70	0xcccb0000
0x73	0xcccc0000

Seitentabelle 2D

Index	Adr.
0x32	0xdddd0000
0x33	0xddde0000
0x70	0xdddf0000
0x73	0xddd10000

a) Bestimmen Sie die **reale Adresse** zur logischen Adresse `0x73703233`. Geben Sie hierbei die zur Bestimmung notwendigen Zwischenschritte stichpunktartig an! (5 Punkte)

b) Nehmen Sie an, dass alle gültigen Seitentabelleneinträge oben abgebildet sind. Was würde in diesem Fall mit einem Prozess passieren, der schreibend auf die Adresse `0xFFFFFFFF` zugreift und wieso? (2 Punkte)